THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Application of	
Inventors: Jose German Rivera et al.	: Confirmation No. 2936
	:
U.S. Patent Application No. 10/786,843	: Group Art Unit: 2192
	:
Filed: February 25, 2004	: Examiner: Zheng WEI
For: METHOD AND APPARATUS FOR MONITORING COMPUTER SOFTWARE	

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Attn: BOARD OF PATENT APPEALS AND INTERFERENCES

BRIEF ON APPEAL

This brief is in furtherance of the Notice of Appeal, filed in this case on September 25, 2008.

The fees required under § 1.17(f) and any required petition for extension of time for filing this brief and fees therefore, are dealt with in the accompanying TRANSMITTAL OF APPEAL BRIEF.

TABLE OF CONTENTS

I. Real Party in Interest

The real party in interest is Hewlett-Packard Development Company, L.P., a limited partnership established under the laws of the State of Texas and having a principal place of business at 20555 S.H. 249 Houston, TX 77070, U.S.A. (hereinafter "HPDC"). HPDC is a Texas limited partnership and is a wholly-owned affiliate of Hewlett-Packard Company, a Delaware Corporation, headquartered in Palo Alto, CA. The general or managing partner of HPDC is HPQ Holdings, LLC.

II. Related Appeals and Interferences

There are no other appeals or interferences that will directly affect, or be directly affected by, or have a bearing on the Board's decision in this appeal.

III. Status of Claims

A. Total Number of Claims in Application

There is a total of 38 claims in the application, which are identified as claims 1, 4-11, 14-21, 24-31, 34-41, and 44-49.

B. Status of all the Claims

Claims 1, 4-11, 14-21, 24-31, 34-41, and 44-49 are pending.

Claims 1, 4-11, 14-21, 24-31, 34-41, and 44-49 are rejected.

Claims 2-3, 12-13, 22-23, 32-33, and 42-43 are cancelled.

Claims allowed – None.

C. Claims on Appeal

Claims on appeal are claims 1, 4-11, 14-21, 24-31, 34-41, and 44-49.

IV. Status of Amendments

There are no outstanding un-entered amendments before the Examiner.

V. Summary of Claimed Subject Matter

The present invention relates generally to a method for monitoring computer software.

Claim 1

Independent claim 1 recites a method for monitoring computer software comprising:

receiving an assertion from an executing process, wherein the executing process is integral to an operating system and wherein receiving an assertion comprises: (See Instant specification in at least paragraphs 10, 12-15, 23-24, 40-43 and FIG. 1, element 5, FIG. 2, element 35, FIG. 8, elements 130, 135, FIG. 9, elements 130, 270, FIG. 12, element 525, FIG. 13, elements 620, 525)

receiving an assertion request; (See Instant specification in at least paragraphs 10, 12-15, 23-24, 40-43 and FIG. 1, element 5, FIG. 2, element 35, FIG. 8, element 130, 135, FIG. 9, elements 130, 270, FIG. 12, element 525, FIG. 13, elements 620, 525)

performing at least one of:

recognizing an assertion request type corresponding to the assertion request; or (See Instant specification in at least paragraphs 13, 15, 16, 23, 25, 28, 41, 43, and FIG. 2, element 40, FIG. 8, elements 135, 150, FIG. 9, elements 270, 285, FIG. 10, element 370, FIG. 12, element 525, FIG. 13, element 530)

determining a component that sourced the assertion request; and (See Instant specification in at least paragraphs 14, 15, 16, 24,

26, 28, 42, 43, and FIG. 2, element 75, FIG. 8, elements 135, 150, FIG. 9, element 285, FIG. 12, element 525, FIG. 13, elements 525, 530)

accepting the assertion request for at least one of:

an assertion request of an enabled recognized assertion request type; or (See Instant specification in at least paragraphs 13, 14, 15, 16, 24, 25, 28, 41, 43, and FIG. 2, elements 45, 50, FIG. 3, FIG. 8, elements 135, 150, FIG. 9, elements 285, 315, FIG. 13, element 530)

an assertion request of a determined component which has assertion requests enabled; (See Instant specification in at least paragraphs 13, 14, 15, 16, 26, 27, 28, 42, 43, and FIG. 2, elements 70, 50, FIG. 3, FIG. 8, elements 135, 150, FIG. 9, elements 285, 315, FIG. 13, element 530)

recording the assertion when the assertion is violated; and (See Instant specification in at least paragraphs 10, 12, 13, 14, 17, 18, 23, 29, 30, 31, 37, 40, 41, 42, 43, 44, 45, 46, and FIG. 1, element 10, FIG. 4, element 90, FIG. 5, elements 95, 100, FIG. 8, element 150, FIG. 12, element 530, FIG. 13, element 530, FIG. 14, element 530)

allowing the executing process to continue execution (See Instant specification in at least paragraphs 10, 12, 13, 14, 40, 41, 42, 43, 45, and FIG. 1, element 15).

Claim 6

Dependent claim 6 recites the method of claim 1 wherein recording the assertion comprises writing information regarding the assertion violation to a circular buffer (See Instant specification in at least paragraph 19 and FIG. 5, element 100, FIG. 10, element 400, FIG. 14, element 517).

Claim 11

Independent claim 11 recites an apparatus for monitoring computer software comprising:

a memory comprising (See Instant specification in at least paragraphs 36-39, and FIG. 12, element 505):

an assertion receiver arranged to receive an assertion from an executing process, wherein the executing process is integral to an operating system and wherein the assertion receiver comprises (See Instant specification in at least paragraph 37 and FIG. 12, element 525):

an assertion request receiver arranged to receive an assertion request; and (See Instant specification in at least paragraphs 10, 12-15, 23-24, 40-43, and FIG. 9, element 270)

an assertion accept determination unit arranged to recognize an assertion type and generate an accept assertion signal for at least one of (See Instant specification in at least paragraphs 24-28, and FIG. 9, element 285):

an enabled recognized assertion type; or (See Instant specification in at least paragraph 24-28, and FIG. 9, element 320)

an enabled determined component; and (See Instant specification in at least paragraph 28, and FIG. 9, element 330)

an assertion recorder arranged to record the assertion when the assertion is violated (See Instant specification in at least paragraph 37, 40-46, and FIG. 10, and FIG. 12, element 530, FIG. 13, element 530, FIG. 14, element 530).

Claim 16

Dependent claim 16 recites the apparatus of claim 11 wherein the assertion recorder comprises:

an information interface arranged to receive assertion violation data; and (See Instant specification in at least paragraph 29-31, and FIG. 10, element 370)

a buffer manager arranged to convey the assertion violation data to a circular buffer (See Instant specification in at least paragraph 19 and FIG. 5, element 100, FIG. 10, elements, 380, 400, 405, FIG. 14, element 517).

Claim 21

Independent claim 21 recites a computer software monitoring system comprising:

memory capable of storing instructions (See Instant specification in at least paragraphs 36-39, and FIG. 12, element 505);

processor capable of executing instructions stored in the memory (See Instant specification in at least paragraphs 36 and 38-39, and FIG. 12, element 500); and

software monitor instruction sequence that, when executed by the processor, minimally causes the processor to (See Instant specification in at least paragraphs 36-49, and FIG. 12, element 520):

receive an assertion from an executing process, wherein the executing process is integral to an operating system and wherein the software monitor instruction sequence comprises an assertion receiver instruction sequence that, when executed by the processor, minimally causes the processor to receive an assertion by minimally causing the processor to (See Instant specification in at least paragraphs 10, 12-15, 23-24, 40-43 and FIG. 1, element 5, FIG. 2, element 35, FIG. 8, elements 130, 135, FIG. 9, elements 130, 270, FIG. 12, element 525, FIG. 13, elements 620, 525):

receive an assertion request (See Instant specification in at least paragraphs 10, 12-15, 23-24, 40-43 and FIG. 1, element 5, FIG. 2, element 35, FIG. 8, element 130, 135, FIG. 9, elements 130, 270, FIG. 12, element 525, FIG. 13, elements 620, 525);

perform at least one of:

recognize a type for the assertion request; or (See Instant specification in at least paragraphs 13, 15, 16, 23, 25, 28, 41, 43, and FIG. 2, element 40, FIG. 8, elements 135, 150, FIG. 9, elements 270, 285, FIG. 10, element 370, FIG. 12, element 525, FIG. 13, element 530)

determine a component that sourced the assertion request; and

accept the assertion request for at least one of: (See Instant specification in at least paragraphs 14, 15, 16, 24, 26, 28, 42, 43, and FIG. 2, element

75, FIG. 8, elements 135, 150, FIG. 9, element 285, FIG. 12, element 525, FIG. 13, elements 525, 530)

an enabled recognized assertion request type; or (See Instant specification in at least paragraphs 13, 14, 15, 16, 24, 25, 28, 41, 43, and FIG. 2, elements 45, 50, FIG. 3, FIG. 8, elements 135, 150, FIG. 9, elements 285, 315, FIG. 13, element 530)

an enabled determined component, (See Instant specification in at least paragraphs 13, 14, 15, 16, 26, 27, 28, 42, 43, and FIG. 2, elements 70, 50, FIG. 3, FIG. 8, elements 135, 150, FIG. 9, elements 285, 315, FIG. 13, element 530)

record the assertion, and (See Instant specification in at least paragraphs 10, 12, 13, 14, 17, 18, 23, 29, 30, 31, 37, 40, 41, 42, 43, 44, 45, 46, and FIG. 1, element 10, FIG. 4, element 90, FIG. 5, elements 95, 100, FIG. 8, element 150, FIG. 12, element 530, FIG. 13, element 530, FIG. 14, element 530)

allow the executing process to continue execution (See Instant specification in at least paragraphs 10, 12, 13, 14, 40, 41, 42, 43, 45, and FIG. 1, element 15).

Claim 26

Dependent claim 26 recites the computer software monitoring system of Claim 21 wherein the software monitor instruction sequence comprises an assertion recorder instruction sequence that, when executed by the processor, minimally causes the processor to record an assertion by minimally causing the processor to write information regarding the assertion to a circular buffer (See Instant specification in at

least paragraph 19 and FIG. 5, element 100, FIG. 10, element 400, FIG. 14, element 517).

Claim 31

Independent claim 31 recites a computer-readable medium having computer-executable instructions for performing a method for monitoring computer software, the instructions comprising modules for:

receiving an assertion from an executing process, wherein the executing process is integral to an operating system and wherein the receiving an assertion module comprises modules for: (See Instant specification in at least paragraphs 10, 12-15, 23-24, 40-43 and FIG. 1, element 5, FIG. 2, element 35, FIG. 8, elements 130, 135, FIG. 9, elements 130, 270, FIG. 12, element 525, FIG. 13, elements 620, 525)

receiving an assertion request; (See Instant specification in at least paragraphs 10, 12-15, 23-24, 40-43 and FIG. 1, element 5, FIG. 2, element 35, FIG. 8, element 130, 135, FIG. 9, elements 130, 270, FIG. 12, element 525, FIG. 13, elements 620, 525)

performing at least one of:

recognizing an assertion request type corresponding to the assertion request; or (See Instant specification in at least paragraphs 13, 15, 16, 23, 25, 28, 41, 43, and FIG. 2, element 40, FIG. 8, elements 135, 150, FIG. 9, elements 270, 285, FIG. 10, element 370, FIG. 12, element 525, FIG. 13, element 530)

determining a component that sourced the assertion request; and (See Instant specification in at least paragraphs 14, 15, 16, 24, 26, 28, 42, 43, and

FIG. 2, element 75, FIG. 8, elements 135, 150, FIG. 9, element 285, FIG. 12, element 525, FIG. 13, elements 525, 530)

accepting the assertion request for at least one of:

an assertion request of an enabled recognized assertion request type; or (See Instant specification in at least paragraphs 13, 14, 15, 16, 24, 25, 28, 41, 43, and FIG. 2, elements 45, 50, FIG. 3, FIG. 8, elements 135, 150, FIG. 9, elements 285, 315, FIG. 13, element 530)

an assertion request of a determined component which has assertion requests enabled; (See Instant specification in at least paragraphs 13, 14, 15, 16, 26, 27, 28, 42, 43, and FIG. 2, elements 70, 50, FIG. 3, FIG. 8, elements 135, 150, FIG. 9, elements 285, 315, FIG. 13, element 530)

recording the assertion; and (See Instant specification in at least paragraphs 10, 12, 13, 14, 17, 18, 23, 29, 30, 31, 37, 40, 41, 42, 43, 44, 45, 46, and FIG. 1, element 10, FIG. 4, element 90, FIG. 5, elements 95, 100, FIG. 8, element 150, FIG. 12, element 530, FIG. 13, element 530, FIG. 14, element 530)

allowing the executing process to continue execution (See Instant specification in at least paragraphs 10, 12, 13, 14, 40, 41, 42, 43, 45, and FIG. 1, element 15).

Claim 36

Dependent claim 36 recites the computer-readable medium of claim 31 wherein the recording the assertion module comprises a module for writing information regarding the assertion to a circular buffer (See Instant specification in at least paragraph 19 and FIG. 5, element 100, FIG. 10, element 400, FIG. 14, element 517).

Claim 41

Independent claim 41 recites an apparatus for monitoring computer software comprising:

means for detecting an assertion from an executing process, wherein the executing process is integral to an operating system and wherein the means for detecting comprises (See Instant specification in at least paragraphs 10, 12-15, 23-24, 40-43 and FIG. 1, element 5, FIG. 2, element 35, FIG. 8, elements 130, 135, FIG. 9, elements 130, 270, FIG. 12, element 525, FIG. 13, elements 620, 525):

means for ascertaining at least one of:

a type of an assertion request; or (See Instant specification in at least paragraphs 13, 15, 16, 23, 25, 28, 41, 43, and FIG. 2, element 40, FIG. 8, elements 135, 150, FIG. 9, elements 270, 285, FIG. 10, element 370, FIG. 12, element 525, FIG. 13, element 530)

a component that sourced an assertion request; and (See Instant specification in at least paragraphs 14, 15, 16, 24, 26, 28, 42, 43, and FIG. 2, element 75, FIG. 8, elements 135, 150, FIG. 9, element 285, FIG. 12, element 525, FIG. 13, elements 525, 530)

means for ignoring the assertion request for at least one of:

a non-enabled ascertained assertion request type; or (See Instant specification in at least paragraphs 13, 14, 15, 16, 24, 25, 28, 41, 43, and FIG. 2, elements 45, 50, FIG. 3, FIG. 8, elements 135, 150, FIG. 9, elements 285, 315, FIG. 13, element 530)

a non-enabled ascertained component; (See Instant specification in at least paragraphs 13, 14, 15, 16, 26, 27, 28, 42, 43, and FIG. 2, elements 70, 50, FIG. 3, FIG. 8, elements 135, 150, FIG. 9, elements 285, 315, FIG. 13, element 530)

means for recording information pertaining to the assertion when it is violated; and (See Instant specification in at least paragraphs 10, 12, 13, 14, 17, 18, 23, 29, 30, 31, 37, 40, 41, 42, 43, 44, 45, 46, and FIG. 1, element 10, FIG. 4, element 90, FIG. 5, elements 95, 100, FIG. 8, element 150, FIG. 12, element 530, FIG. 13, element 530, FIG. 14, element 530)

means for allowing the executing process to continue execution (See Instant specification in at least paragraphs 10, 12, 13, 14, 40, 41, 42, 43, 45, and FIG. 1, element 15).

VI. Grounds of Rejection to be Reviewed on Appeal

A. The issue is whether claims 1, 4, 5, 7-11, 14, 15, 17-21, 24, 25, 27-35, 37-41, and 44-49 are unpatentable under 35 U.S.C 103(a) as being obvious over *Williams (Mickey Williams, "Microsoft Visual C# .NET")* in view of *GNU "(The GNU C Library, Section 'Explicitly Checking Internal Consistency')"* and further in view of *PHP "(PHP Manual, Section 'assert_options')"*.

B. The issue is whether claims 6, 16, 26, and 36 are unpatentable under 35 U.S.C. 103(a) as being obvious over *Williams* in view of *GNU*, *PHP*, and *Cantrill* (U.S. 7,146,473).

VII. Argument

A. Was the PTO correct in rejecting claims 1, 4, 5, 7-11, 14, 15, 17-21, 24, 25, 27-35, 37-41, and 44-49 under 35 U.S.C. 103(a) as being obvious over *Williams* in view of *GNU* and further in view of *PHP*?

Claim 1

The rejection of claims 1, 4, 5, 7-11, 14, 15, 17-21, 24, 25, 27-35, 37-41, and 44-49 as being unpatentable over *Williams* in view of *GNU* and further in view of *PHP* is hereby traversed. Claim 1 is patentable over *Williams* in view of *GNU* and *PHP* because the references, singly or in combination, fail to disclose or suggest every feature of claim 1.

Claim 1

Reply to Response to Arguments

At the outset, Appellants reply to the PTO assertions made in the Response to Arguments section from page 2 through page 5.

1. Neither *Williams* nor *GNU* disclose or suggest "receiving an assertion from an executing process, wherein the executing process is integral to an operating system"

The PTO asserts that "it is the [sic] *Williams* that discloses said limitation as the Applicants argued" and that the "cited portion of *GNU* (as the secondary reference) is used to teach the limitation of 'the executing process is integral to an operating system' that is not explicitly disclosed by the primary reference *Williams*." Final Office Action mailed July 25, 2008 at page 2, paragraph bridging over to page 3. This is incorrect because neither *Williams* nor *GNU* appear to disclose or suggest "receiving an assertion from an executing process, wherein the executing process is integral to an operating system" as claimed in claim 1.

The PTO asserted in the FOA that *Williams* disclosed "receiving an assertion from an executing process;" however, the PTO also admitted that *Williams* did not

disclose that "the executing process is integral to an operating system." Therefore, the claim 1 feature of receiving an assertion from an executing process wherein the executing process is integral to an operating system is not met by *Williams*. For at least this reason, reversal of the rejection is respectfully requested.

Further, *GNU* fails to disclose or suggest a method where an assertion is received from an executing process which is integral to an operating system. *GNU* appears directed to user applications and not to an operating system process. For example, *GNU* states "[w]hen you're writing a program, it's often a good idea to put in checks" and differentiates the program from an operating system stating that "[s]ometimes the 'impossible' condition you want to check for is an error return from an operating system function." *GNU* at page 1, second paragraph and page 2, third paragraph. Thus, *GNU* describes, at most, checking an error condition in a program which is not an operating system. Based on at least the foregoing, *GNU* appears to not be directed to the situation of receiving an assertion from an executing process which is integral to an operating system as claimed in claim 1. For at least this reason, reversal of the rejection is respectfully requested.

2. *GNU* fails to disclose receiving an assertion from an executing process integral to an operating system

The PTO asserts that the "*GNU* reference is used to teach the limitation of 'the executing process is integral to an operating system.'" The PTO is in error and has not responded to Appellants arguments showing that *GNU* fails to disclose receiving an assertion from an executing process integral to an operating system. *GNU* appears to describe using an assert macro to check for "an error return from an operating system function" and not receipt of an assertion from an operating system. The error return, as described, causes the program assertion to fail and abort, but there is no disclosure of an assertion from the operating system.

Further, *GNU* states that on an assertion being given invalid input, a program will abort. See page 2, penultimate paragraph ("your program should not abort when given invalid input, as assert would do").

Based on at least each of the foregoing reasons, reversal of the rejection is respectfully requested.

3. *GNU* explicitly teaches a program aborting without proceeding to continue execution as a result of an assert occurrence

The PTO asserts that the *GNU* reference is used to teach the limitation of "'the executing process is integral to an operating system' that is not explicitly disclosed by the primary reference *Williams*." FOA at page 4, second paragraph. The PTO attempts to ignore the plain teaching of *GNU* with respect to aborting programs without continuing execution as a result of an assertion. That is, *GNU* explicitly states that through the use of the assert macro, execution of the program is aborted on an evaluation of an expression to false (zero). See *GNU* at page 1, final three paragraphs. Thus, *GNU* fails to disclose receiving an assertion and allowing the executing process to continue execution and in fact *GNU* positively discloses the opposite whereby a program receiving an assertion aborts without continuing execution. For at least this reason, reversal of the rejection is respectfully requested. *Williams* and the button labeled "Ignore" are discussed below.

4. The PTO has failed to set forth a prima facie case of obviousness

The PTO fails to overcome Appellants' showing that the PTO has failed to make out a prima facie case of obviousness with respect to the combination of *Williams* and *GNU*, i.e., the PTO has failed to articulate a reasonable rationale for combining *Williams* and *GNU*. Appellants set forth at least one rationale identifying why a person of ordinary skill in the art would **not** combine *GNU* with *Williams* and in doing so Appellants relied on the descriptive text of the Instant Specification and additionally supporting statements from *GNU*.

The PTO assertion that "claim language does not limit and/or require to meet such conditions" is inapplicable to the issue of whether the PTO has set forth a prima facie case of obviousness. Appellants relied on both the Instant Specification and the clear disclosure of *GNU* in traversing the asserted combination and the PTO has failed

to overcome Appellants showing. For at least this reason, reversal of the rejection is respectfully requested.

5. *PHP* fails to disclose recognizing an assertion request type

The PTO fails to overcome Appellants' showing that *PHP* appears to describe the setting/getting of various assert flags related to control options of an assert. The PTO has failed to articulate how setting/getting various assert flags corresponds to recognition of an assertion request type as claimed in claim 1. The bald statement by the PTO that the "'assert request type' active in *PHP* does read the claim limitation" fails to provide any rationale or reference to *PHP* supporting the PTO position. The PTO has failed to identify any such disclosure or teaching of an "assert request type" in *PHP*. For at least this reason, reversal of the rejection is respectfully requested.

Appellants cited to the Instant Specification in order to direct the PTO's attention to at least one benefit identified in Appellants' specification related to the recognition of assertion request types.

6. *PHP* fails to disclose a determination of the component that sourced the assertion request

The PTO baldly states without any identified support in *PHP* that "the Examiner's position is that *PHP*'s functions 'assert_option()' / 'assert()' are called/queried by the caller/process and the functions' return value automatically returns the original caller (assertion request) and further can identify the component that source the assertion request." FOA at page 5, first paragraph. The PTO fails to identify any support in *PHP* for the foregoing. Based on the description of *PHP*, assert_options enables the setting/getting of various assert flags. *PHP* at page 1. *PHP*, and specifically the assert_options portion relied on, fails to disclose or suggest a determination of the component that sourced an assertion request. For at least this reason, reversal of the rejection is respectfully requested.

Stated Rejections from the Final Office Action

As set forth by Appellants in the prior response submitted April 10, 2008 and the above discussion, the PTO agrees that *Williams* fails to disclose or suggest "receiving an assertion from an executing process, wherein the executing process is integral to an operating system" as claimed claim 1. The PTO attempts to rely on *GNU* to overcome the admitted deficiency of *Williams*. This is incorrect and fails to overcome the deficiency of *Williams*.

First, the PTO asserts that *GNU* describes "using assert method in the operating system and report execution error (see for example, p. 1, section 'Explicitly Checking Internal Consistency', first paragraph and p.2 third paragraph, 'check for an error return from an operating system function')." See FOA at page 6, lines 16-19. This is incorrect. The first PTO-identified portion of *GNU*, provided for ease of reference, states:

> When you're writing a program, it's often a good idea to put in checks at strategic places for "impossible" errors or violations of basic assumptions. These kinds of checks are helpful in debugging problems with the interfaces between different parts of the program, for example.
>
> *GNU* at page 1, paragraph 1 of Section "Explicitly Checking Internal Consistency."

The above portion of *GNU* fails to disclose or suggest using an assert method **in an operating system** and reporting an execution error as claimed in claim 1. The cited portion of *GNU* appears to disclose using checks in programs for debugging problems without disclosing receiving an assertion from an executing process **integral to an operating system**, recording the assertion, and allowing the executing process (which is integral to the operating system) to continue execution. For at least this reason, reversal of the rejection is respectfully requested.

The second PTO-identified portion of *GNU*, provided for ease of reference, states:

> Sometimes the "impossible" condition you want to check for is an error return from an operating system function. Then it is useful to display not only where the program crashes, but also what error was returned. The assert_perror macro makes this easy.
>
> *GNU* at page 2, paragraph 3 of Section "Explicitly Checking Internal Consistency."

The above portion of *GNU* fails to disclose or suggest receiving an assertion from an executing process **integral to an operating system** and reporting an execution error as claimed in claim 1. The cited portion of *GNU* appears to disclose checking for an 'impossible' condition **in a program** which receives an error return from an operating system function. That is, the cited portion of *GNU* appears to disclose use of the "assert_perror" macro in a program in order to check for an error returned from an operating system and not use of the macro in the operating system. For at least this reason, reversal of the rejection is respectfully requested.

Second, *GNU* appears to disclose that a program aborts without proceeding to continue execution as a result of an assert occurrence. *GNU* at page 2, paragraph 11 of Section "Explicitly Checking Internal Consistency" - "your program should not abort when given invalid input, as assert would do." That is, *GNU* discloses aborting execution upon occurrence and not the claimed receiving an assertion, recording the assertion, and allowing the executing process to continue execution. Additionally, see *GNU* at page 1, paragraph 3, which states that the assert macro "provides a convenient way to **abort the program** while printing a message about where in the program the error was detected." (emphasis added). Thus, the execution is aborted and execution does not continue and a combination of *Williams* and *GNU* fails to render obvious the claimed subject matter. For at least this reason, reversal of the rejection is respectfully requested.

Third, the PTO fails to set forth a prima facie case of obviousness and asserts that a person of ordinary skill in the art at the time of the present invention would have been motivated to combine *GNU* with *Williams* in order to "display all the error information and further help to debug the problem as suggested by *GNU*." OA at page 3, final three lines of the page. This is incorrect and fails to overcome Applicant's

description that release code "is typically devoid of assertions because the assertions cause performance degradation." See Instant Specification at paragraph 8. Contrary to the PTO's assertion, *GNU* buttresses this fact at page 1, paragraph 5, reciting that the consistency checks may "make the program significantly slower." Further, Applicants' description recites that "[a]borting . . . continuously running software [i.e., an operating system] results in a system crash, which is an unacceptable artifact of a violated assertion." See Instant Specification at paragraph 8. Thus, the aborting of a program as set forth in *GNU* is an unacceptable occurrence with respect to an operating system. This too is supported by *GNU* which states that a "wise user would rather have a program crash, visibly, than have it return nonsense without indicating anything might be wrong." See *GNU* at page 1, paragraph 5. For at least this reason, the PTO has failed to articulate a reasonable rationale for combining the *Williams* and *GNU* as asserted and a prima facie case of obviousness has not been set forth. For at least this reason, reversal of the rejection is respectfully requested.

Fourth, the PTO admits that neither *Williams* nor *GNU* discloses recognizing an assertion request type corresponding to the assertion request. The PTO asserts that *PHP* cures the admitted deficiency of *Williams* and *GNU*. This is also incorrect.

The cited portion of *PHP* appears to describe options useable to control an assertion and fails to disclose an assertion request type as in the present claimed subject matter. The cited portion of *PHP* describes the setting/getting of "various assert flags" related to control options of an assert. Thus, *PHP* does not appear to identify a need nor a disclosure of recognizing an assertion request type as claimed.

Further, as set forth in paragraph 13 of the Instant Specification, being able to differentiate between assertion request types may be beneficial:

> [b]ecause different assertion types generally require different amounts of processor resources, the ability to enable only specific types of assertions allows a programmer to better manage the trade-off between the usefulness of a particular type of assertion and its associated cost (in required processor resources)."
>
> For at least this reason, reversal of the rejection is respectfully requested.

Fifth, the PTO asserts that *PHP* discloses "determining a component that sourced the assertion request" in p. 1, Table 1. Assert Options. This is incorrect because *PHP* fails to disclose a determination of the component that sourced the assertion request, with or without determining whether the component has assertion requests enabled. *PHP* appears to disclose, as set forth above, setting and getting various flags related to the assert without relation to a particular component and without disclosing determining the component which sourced the assertion request. For at least this reason, reversal of the rejection is respectfully requested.

Sixth, the PTO asserts that *Williams* discloses "allowing the executing process to continue execution." This is believed incorrect because although Figure 9-3 depicts a dialog box having an Ignore button (among abort and retry buttons) there is no supporting description related to operation of the assert method as a result of a user actuating the ignore button (or the abort or retry buttons). The description at page 10 of *Williams* further states only that dialog box displayed as a result of encountering an assertion failure would be "similar to the one shown in Figure 9-3." Thus, *Williams* fails to definitively disclose the contents of the displayed dialog box and, potentially more importantly, fails to disclose what happens as a result of activating one of the displayed buttons. For at least this reason, reversal of the rejection is respectfully requested.

Further, combining *GNU* with *Williams arguendo* appears to suggest that activating any of the displayed buttons would result in "abort[ing] the program while printing a message about where in the program the error was detected." *GNU* at p. 1, paragraph 3. See also *GNU* at p. 1, paragraph 7 "If it is false (zero), assert aborts the program . . . after printing a message." Thus, the asserted combination of *GNU* and *Williams* appears to suggest aborting a program after assertion receipt and not "allowing the executing process to continue execution" as claimed in claim 1. For at least this reason, reversal of the rejection is respectfully requested.

Based on each of the foregoing reasons, amended claim 1 is patentable over *Williams*, alone or in combination with *GNU* and *PHP*, and the rejection is respectfully requested to be reversed.

Claims 4, 5, 7-11, 14, 15, 17-21, 24, 25, 27-35, 37-41, and 44-49

Claims 4, 5, 7-11, 14, 15, 17-21, 24, 25, 27-35, 37-41, and 44-49 depend, either directly or indirectly, from claims 1, 11, 21, 31, and 41, include further features, and are patentable over *Williams* in view of *GNU* and further in view of *PHP* for at least the reasons advanced above with respect to claim 1. The rejection of claims 4, 5, 7-11, 14, 15, 17-21, 24, 25, 27-35, 37-41, and 44-49 should be reversed.

VIII. Argument

B. Was the PTO correct in rejecting claims 6, 16, 26, and 36 under 35 U.S.C. 103(a) as being obvious over *Williams* in view of *GNU*, *PHP*, and *Cantrill*?

Claim 6

The rejection of claims 6, 16, 26, and 36 as being unpatentable over *Williams* in view of *GNU*, *PHP*, and *Cantrill* is hereby traversed. Claims 6, 16, 26, and 36 depend, either directly or indirectly, from claims 1, 11, 21, and 31, include further features, and are patentable over *Williams* in view of *GNU*, *PHP*, and *Cantrill* for at least the reasons advanced above with respect to claims 1, 11, 21, and 31, respectively. The rejection of claims 6, 16, 26, and 36 is respectfully requested to be reversed.

IX. Conclusion

Each of the PTO's rejections has been traversed. Appellant respectfully submits that all claims on appeal are considered patentable over the applied art of record. Accordingly, reversal of the PTO's Final Rejection is believed appropriate and courteously solicited.

If for any reason this Appeal Brief is found to be incomplete, or if at any time it appears that a telephone conference with counsel would help advance prosecution, please telephone the undersigned, Appellant's attorney of record.

To the extent necessary, a petition for an extension of time under 37 C.F.R. 1.136 is hereby made. Please charge any shortage in fees due in connection with the filing of this paper, including extension of time fees, to Deposit Account 08-2025 and please credit any excess fees to such deposit account.

Reversal of the rejection is in order.

Respectfully submitted,
Jose German Rivera et al.

By: /Randy A. Noranbrock/
Randy A. Noranbrock
Registration No. 42,940
Telephone: 703-684-1111

HEWLETT-PACKARD COMPANY
IP Administration
Legal Department, M/S 35
P.O. Box 272400
Fort Collins, CO 80528-9599
Telephone: 970-898-7057
Facsimile: 281-926-7212
Date: **November 25, 2008**
RAN/bjs

X. Claims Appendix

1. A method for monitoring computer software comprising:

receiving an assertion from an executing process, wherein the executing process is integral to an operating system and wherein receiving an assertion comprises:

receiving an assertion request;

performing at least one of:

recognizing an assertion request type corresponding to the assertion request; or

determining a component that sourced the assertion request; and

accepting the assertion request for at least one of:

an assertion request of an enabled recognized assertion request type; or

an assertion request of a determined component which has assertion requests enabled;

recording the assertion when the assertion is violated; and

allowing the executing process to continue execution.

4. The method of Claim 1 wherein recording the assertion comprises recording a datum that includes at least one of:

type of assertion,

sequence number of the assertion,

time at which the assertion occurred,

identification of processor that produced the assertion,

identification of process that produced the assertion,

identification of the thread that produced the assertion,

text of the assertion,

stack trace,

source line containing the assertion, or

file name of the source containing the code that generated the assertion.

5. The method of Claim 1 wherein recording the assertion comprises writing information regarding the assertion violation to a computer readable medium.

6. The method of Claim 1 wherein recording the assertion comprises writing information regarding the assertion violation to a circular buffer.

7. The method of Claim 1 further comprising:

accepting a command from at least one of a control console and a network connection; and

updating an enable condition for an assertion class according to the command.

8. The method of Claim 1 further comprising generating an error report according to the recorded assertion.

9. The method of Claim 8 further comprising dispatching the error report to a real-time assertion monitor.

10. The method of Claim 8 wherein generating an error report comprises:

retrieving an assertion violation parameter including at least one of:

type of assertion,

sequence number of the assertion,

time at which the assertion occurred,

identification of processor that produced the assertion,

identification of process that produced the assertion,

identification of the thread that produced the assertion,

text of the assertion,

stack trace,

source line containing the assertion, or

file name of the source containing the code that generated the assertion; and

generating a report file comprising page description statements according to the assertion parameter.

11. An apparatus for monitoring computer software comprising:

a memory comprising:

an assertion receiver arranged to receive an assertion from an executing process, wherein the executing process is integral to an operating system and wherein the assertion receiver comprises:

an assertion request receiver arranged to receive an assertion request; and

an assertion accept determination unit arranged to recognize an assertion type and generate an accept assertion signal for at least one of:

an enabled recognized assertion type; or

an enabled determined component; and

an assertion recorder arranged to record the assertion when the assertion is violated.

14. The apparatus of Claim 11 wherein the assertion recorder is capable of recording a datum that includes at least one of:

type of assertion,

sequence number of the assertion,

time at which the assertion occurred,

identification of processor that produced the assertion,

identification of process that produced the assertion,

identification of the thread that produced the assertion,

text of the assertion,

stack trace,

source line containing the assertion, or

file name of the source containing the code that generated the assertion.

15. The apparatus of Claim 11 wherein the assertion recorder comprises:

an information interface arranged to receive assertion violation data; and

a media controller arranged to convey the assertion violation data to a computer readable medium.

16. The apparatus of Claim 11 wherein the assertion recorder comprises:

an information interface arranged to receive assertion violation data; and

a buffer manager arranged to convey the assertion violation data to a circular buffer.

17. The apparatus of Claim 11 further comprising:

a command receiver arranged to accept a command from at least one of a control console or a network connection; and

an assertion manager arranged to update an enable condition for an assertion class according to the command.

18. The apparatus of Claim 11 further comprising an error report generator arranged to generate an error report according to the recorded assertion.

19. The apparatus of Claim 18 further comprising a dispatch unit arranged to dispatch an error report to a real-time assertion monitor.

20. The apparatus of Claim 18 wherein the error report generator comprises:

a data retrieval unit that retrieves an assertion violation parameter including at least one of:

type of assertion,

sequence number of the assertion,

time at which the assertion occurred,

identification of processor that produced the assertion,

identification of process that produced the assertion,

identification of the thread that produced the assertion,

text of the assertion,

stack trace,

source line containing the assertion, or

file name of the source containing the code that generated the assertion; and

a report file generator arranged to generate a report file comprising page description statements according to the assertion parameter.

21. A computer software monitoring system comprising:

memory capable of storing instructions;

processor capable of executing instructions stored in the memory; and

software monitor instruction sequence that, when executed by the processor, minimally causes the processor to:

receive an assertion from an executing process, wherein the executing process is integral to an operating system and wherein the software monitor instruction sequence comprises an assertion receiver instruction sequence that, when executed by the processor, minimally causes the processor to receive an assertion by minimally causing the processor to:

receive an assertion request;

perform at least one of:

recognize a type for the assertion request; or

determine a component that sourced the assertion request; and

accept the assertion request for at least one of:

an enabled recognized assertion request type; or

an enabled determined component,

record the assertion, and

allow the executing process to continue execution.

24. The computer software monitoring system of Claim 21 wherein the software monitor instruction sequence comprises an assertion recorder instruction sequence that, when executed by the processor, minimally causes the processor to record an assertion by minimally causing the processor to record a datum that includes at least one of:

type of assertion,

sequence number of the assertion,

time at which the assertion occurred,

identification of processor that produced the assertion,

identification of process that produced the assertion,

identification of the thread that produced the assertion,

text of the assertion,

stack trace,

source line containing the assertion, or

file name of the source containing the code that generated the assertion.

25. The computer software monitoring system of Claim 21 wherein the software monitor instruction sequence comprises an assertion recorder instruction sequence that, when executed by the processor, minimally causes the processor to record an assertion by minimally causing the processor to write information regarding the assertion to a computer readable medium.

26. The computer software monitoring system of Claim 21 wherein the software monitor instruction sequence comprises an assertion recorder instruction sequence that, when executed by the processor, minimally causes the processor to record an assertion by minimally causing the processor to write information regarding the assertion to a circular buffer.

27. The computer software monitoring system of Claim 21 wherein the software monitor instruction sequence further minimally causes the processor to:
accept a command from at least one of a control console or a network connection; and
update an enable condition for an assertion class according to the command.

28. The computer software monitoring system of Claim 21 wherein the software monitor instruction sequence further minimally causes the processor to generate an error report according to the recorded assertion.

29. The computer software monitoring system of Claim 28 wherein the software monitor instruction sequence further minimally causes the processor to dispatch the error report to a real-time assertion monitor.

30. The computer software monitoring system of Claim 28 wherein the software monitor instruction sequence comprises an error report generator instruction sequence that, when executed by the processor, minimally causes the processor to generate an error report by minimally causing the processor to:

retrieve an assertion violation parameter including at least one of:

type of assertion,

sequence number of the assertion,

time at which the assertion occurred,

identification of processor that produced the assertion,

identification of process that produced the assertion,

identification of the thread that produced the assertion,

text of the assertion,

stack trace,

source line containing the assertion, or

file name of the source containing the code that generated the assertion; and

generate a report file comprising page description statements according to the assertion parameter.

31. A computer-readable medium having computer-executable instructions for performing a method for monitoring computer software, the instructions comprising modules for:

receiving an assertion from an executing process, wherein the executing process is integral to an operating system and wherein the receiving an assertion module comprises modules for:

receiving an assertion request;

performing at least one of:

recognizing an assertion request type corresponding to the assertion request; or

determining a component that sourced the assertion request; and

accepting the assertion request for at least one of:

an assertion request of an enabled recognized assertion request type; or

an assertion request of a determined component which has assertion requests enabled;

recording the assertion; and

allowing the executing process to continue execution.

34. The computer-readable medium of Claim 31 wherein the recording the assertion module comprises a module for recording a datum that includes at least one of:

type of assertion,

sequence number of the assertion,

time at which the assertion occurred,

identification of processor that produced the assertion,

identification of process that produced the assertion,

identification of the thread that produced the assertion,

text of the assertion,

stack trace,

source line containing the assertion, or

file name of the source containing the code that generated the assertion.

35. The computer-readable medium of Claim 31 wherein the recording the assertion module comprises a module for writing information regarding the assertion to a computer readable medium.

36. The computer-readable medium of Claim 31 wherein the recording the assertion module comprises a module for writing information regarding the assertion to a circular buffer.

37. The computer-readable medium of Claim 31, the instructions further comprising modules for:

accepting a command from at least one of a control console or a network connection; and

updating an enable condition for an assertion class according to the command.

38. The computer-readable medium of Claim 31, the instructions further comprising a module for generating an error report according to the recorded assertion.

39. The computer-readable medium of Claim 38, the instructions further comprising a module for dispatching the error report to a real-time assertion monitor.

40. The computer-readable medium of Claim 38 wherein dispatching the error report module comprises modules for:

retrieving an assertion violation parameter including at least one of:

type of assertion,

sequence number of the assertion,

time at which the assertion occurred,

identification of processor that produced the assertion,

identification of process that produced the assertion,

identification of the thread that produced the assertion,

text of the assertion,

stack trace,

source line containing the assertion, or

file name of the source containing the code that generated the assertion; and

generating a report file comprising page description statements according to the assertion parameter.

41. An apparatus for monitoring computer software comprising:

means for detecting an assertion from an executing process, wherein the executing process is integral to an operating system and wherein the means for detecting comprises:

means for ascertaining at least one of:

a type of an assertion request; or

a component that sourced an assertion request; and

means for ignoring the assertion request for at least one of:

a non-enabled ascertained assertion request type; or

a non-enabled ascertained component;

means for recording information pertaining to the assertion when it is violated; and

means for allowing the executing process to continue execution.

44. The method of Claim 1 wherein the assertion request type is one of a group of defined assertion macro names.

45. The apparatus of Claim 11 wherein the assertion type is one of a group of defined assertion macro names.

46. The computer software monitoring system of Claim 21 wherein the assertion request type is one of a group of defined assertion macro names.

47. The computer-readable medium of Claim 31 wherein the assertion request type is one of a group of defined assertion macro names.

48. The apparatus for monitoring computer software of Claim 41 wherein the assertion request type is one of a group of defined assertion macro names.

49. A method for monitoring computer software comprising:

receiving an assertion from an executing process, wherein the executing process is integral to an operating system;

recording the assertion when the assertion is violated; and

allowing the executing process to continue execution.

XI. Evidence Appendix

None.

XII. Related Proceedings Appendix

None.